Attorney Docket No.: 06502.0254

United States Patent Application

of

Peter C. Jones

Robert W. Scheifler

and

James H. Waldo

for

Method and System for

Establishing Trust in Downloaded Proxy Code

LAW OFFICES
FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

RELATED APPLICATION

The present application is related to U.S. Provisional Patent Application No. 60/128,406, entitled "RMI Security," filed April 8, 1999, which is relied upon and is incorporated herein by reference.

FIELD OF THE INVENTION

The present invention relates generally to data processing systems and, more particularly, to using downloaded code to provide secure communication between a client and a remote service in a distributed system.

BACKGROUND OF THE INVENTION

Today's distributed systems can be made up of various components, including both hardware and software. Nodes in distributed systems typically communicate via a network such as the Internet. One means of communication between programs in a distributed system is downloading code from one program to another. For example, a client (e.g., a program running on a node in a distributed system) can access a service running on a remote node by downloading code from the remote service.

A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or related to providing access to another user. Examples of services include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system.

In a distributed system implemented using the Java™ programming language, services appear programmatically as objects, with interfaces that define the operations available from the service. In the Java™ programming language, a "class" provides a template for the creation of "objects" (which represent items or instances manipulated by the system) having characteristics of that class. Thus, a class defines the type of an object. Methods associated with a class are generally invoked on the objects of the same class or subclass. The Java™ programming language is described in The Java™ Language Specification by James Gosling, Bill Joy, and Guy Steele, Addison-Wesley, 1996, which is incorporated herein by reference.

In a distributed system that depends on downloaded code, there is the danger that the downloaded code may be untrustworthy. For example, the code could carry a virus or disguise its true source, causing the user of the downloaded code to disclose confidential information to an unknown party. Therefore, it is desirable to enable users in a distributed system to confirm that downloaded code is trustworthy.

SUMMARY OF THE INVENTION

A system consistent with the present invention enables a user in a distributed system to determine whether downloaded code is trustworthy before using the downloaded code to communicate with others in the distributed system. For example, if a client downloads code from a service, the client can verify that both the service and the downloaded code are trustworthy before using the code to communicate with the service. "Trustworthy" code is code that the client knows will enforce the client's security constraints (e.g., mutual authentication, confidentiality, and integrity) when communicating with the service,.

In accordance with one implementation of the present invention, code is downloaded from a server, and a set of constraints to implement secure communication with the server is determined. Secure code is then used to verify that the downloaded code will enforce the set of constraints when the downloaded code is used to communicate with the server.

In accordance with another implementation of the present invention, a first proxy containing code for communication purposes is downloaded, and a second proxy containing code for communication purposes is obtained from the first proxy. A trustworthiness verification routine is used to determine whether the second proxy is trustworthy, and when it has been determined that the second proxy is trustworthy, the second proxy is used to determine whether a server is trustworthy. When it has been determined that the server is trustworthy, a trustworthiness verification routine is requested from the server by using the second proxy and this verification routine is then used to identify the trustworthiness of the first proxy. When it has been determined that the second proxy is trustworthy, that the server is trustworthy, and that the first proxy is trustworthy, the first proxy is used to invoke a method on the server.

BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

Fig. 1 depicts a distributed system suitable for practicing methods and systems consistent with the present invention;

Fig. 2 depicts the logical interaction between a client and a service once the client has downloaded a proxy;

Fig. 3 is a high-level flow chart of an exemplary method for establishing trust in a downloaded proxy in accordance with methods and systems consistent with the present invention; and

Figs. 4A and 4B are flow charts depicting the steps of the verifyProxyTrust method for establishing trust in a downloaded proxy.

DETAILED DESCRIPTION OF THE INVENTION

A distributed system suitable for practicing methods and systems consistent with the present invention can be implemented using the Jini™ architecture. A Jini™ system is built around one or more lookup services that list the services available in the distributed system. When a service joins the network, it uses a process called discovery to locate the lookup service or services for the network. The service registers by passing a proxy object to each lookup service. The proxy object is a Java™ object implementing the interfaces of the corresponding service. When a client (e.g., a program) wishes to use a service, the client interacts with the lookup service and downloads the proxy to facilitate use of the service. The Jini™ architecture is described in more detail in Arnold, The Jini™ Specification, Addison-Wesley (1999).

A system consistent with the present invention enables a program in a distributed system to determine whether downloaded code is trustworthy before using the downloaded code to communicate with other programs or services in the distributed system. For example, if a client downloads code from a service, the client can verify that both the service and the downloaded code are trustworthy before using the code to communicate with the service. "Trustworthy" code is code the client knows will enforce the client's security constraints in communicating with the service.

These constraints can include, for example, integrity, anonymity, mutual authentication, delegation, and confidentiality. The "integrity" constraint ensures that messages will not be tampered with in transit, the "anonymity" constraint permits the identity of the client to stay unknown to the service, and "mutual authentication" refers to the client and service verifying their identities to one another. The "delegation" constraint allows the service to make calls to other computers using the client's identity, and the "confidentiality" constraint ensures that messages are private, e.g., by using encryption.

The constraints attached to a proxy are included in the serialized version of the proxy. As part of RMI system 122, an object, such as a proxy, is converted into a serialized version of itself before being passed. The serialized object contains enough information to enable the recipient to identify and verify the Java™ class from which the contents of the object were saved and to restore the contents to a new instance. Object serialization is explained in the Java™ Object Serialization Specification, available at http://web2.java.sun.com/products//jdk/1.3/docs/guide/serialization/spec/serialTOC.doc.html, which is incorporated herein by reference. When a client trusts the downloaded code to enforce the client's security constraints and trusts the service, the client can trust communicating through the proxy to the service and can trust the service to do what the client asks, and not something harmful.

Figure 1 depicts a distributed system 100 suitable for practicing methods and systems consistent with the present invention. Distributed system 100 includes a plurality of computers 102, 104, and 106, communicating via a network 108. Network 108 may be a local area network, wide area network, or the Internet. Computer 102 includes a memory 110, a secondary storage device 112, a central processing unit (CPU) 114, an input device 116, and a video display 118. Memory 110 includes a client program 142 and Java™ runtime system 120, which includes a Java™ remote method invocation (RMI) system 122.

Computers 104 and 106 may be similarly configured, with computer 104 including memory 124, secondary storage device 126, and CPU 128, and computer 106 including memory 130, secondary storage device 132, and CPU 134. Memory 124 of computer 104 includes a service 136, and memory 130 of computer 106 includes a lookup service 138 that defines the services available in one part of distributed system 100. Lookup service 138 contains one proxy object for each service within that part of distributed system 100. Each "proxy object" corresponds to a service in the distributed system and implements interfaces corresponding to methods available from a service.

For example, if service 136 were a database service, the corresponding proxy object 140 would implement database interfaces such as "find" or "sort." To invoke methods on database service 136, client 142 asks lookup service 138 for a proxy to the database service. In response, lookup service 138 returns proxy object 140 corresponding to database service 136 to the client. Lookup services are explained in greater detail in U.S. Patent Application No.09/044,931, entitled "Dynamic Lookup Service in a Distributed System," which is incorporated herein by reference.

Figure 2 depicts the logical interaction between client program 142 and service 136 once the client has downloaded proxy 140. Proxy 140 implements one or more interfaces 202 corresponding to the methods 204 offered by service 136. Client 142 invokes methods on proxy 140. Proxy 140 processes method invocations, communicates with service 136 to execute the requests of client 142, and returns results. The protocol between proxy 140 and service 136 is not set by the Jini™ system; instead, it is determined by the service and its proxy. This enables a client to communicate with a service it may have never seen before without knowing or caring how the service works.

In this way, the Jini™ architecture relies on downloaded code to provide services in a dynamic and flexible way. Once a client downloads a proxy object, the client invokes methods on the proxy to communicate with the remote service. However, the proxy may be implemented using code that the client does not trust. Before sharing any critical data with the service, the client needs to establish trust in the proxy, the service, or both.

Figure 3 is a high-level flow chart of an exemplary method for establishing trust in a downloaded proxy in accordance with methods and systems consistent with the present invention. The exemplary method enables a client to use code that can be verified locally to communicate with a remote service. Using the locally-verified code, the client verifies the remote service and then asks the service whether the service trusts the downloaded proxy. If the client has verified the service and the service trusts the proxy, then the client has established transitive trust in the proxy.

With reference to figure 3, the exemplary method begins when the client downloads the proxy, P1, from a lookup service (step 302). The client then asks P1 for a second proxy to the service, P2, by invoking a method on P1 corresponding to a method on the service (step 304). The client examines the code in P2 to verify that P2 uses only trusted code (step 306). If the client verifies P2 (step 307), then the client uses P2 to authenticate the service by invoking a method on P2 (step 308). If the service is authenticated (step 309), the client then uses P2 to ask the service if the service trusts P1's code, by invoking a method on P2 to obtain a proxy verifier from the service, and then passing P1 to the proxy verifier (step 310). If the service trusts P1 (step 311), then the client can trust P1 (step 312). In this manner, the client uses local, trusted code to establish trust in a downloaded proxy.

The VerifyProxyTrust method

The exemplary method described above will now be explained using the Java™ RMI Security Subsystem, part of RMI system 122. The Security Subsystem defines classes and interfaces that ensure secure communication between remote objects. One class provided by the RMI Security Subsystem is the Security class, which includes the verifyProxyTrust method. The verifyProxyTrust method establishes trust in downloaded code and is defined as follows:

```
public static void verifyProxyTrust (
        Object proxy,
        SecurityConstraints constraints)
throws RemoteException;
```

The verifyProxyTrust method in RMI system 122 is called by client 142 once the client downloads proxy 140, before the client makes any other use of the proxy. As described below, the method will establish trust in the proxy by confirming that the proxy will correctly implement the RemoteSecurity interface, part of the Security class.

When a remote service instantiates its proxy, the service can include the RemoteSecurity interface, which provides methods enabling a client to attach security constraints to the proxy or to query the service to determine the service's security constraints. The RemoteSecurity interface is defined as follows:

```
public interface RemoteSecurity {
        RemoteSecurity setClientConstraints (
                        SecurityConstraints constraints);
        SecurityConstraints getClientConstraints();
        SecurityConstraints getServerConstraints (String name,
                    Class[] parameterTypes);
              throws NoSuchMethodException, RemoteException;
        boolean equalsIgnoreConstraints(Object obj);
    }
```

The setClientConstraints method allows a client to make a copy of the proxy with a new set of constraints selected by the client. The getClientConstraints method returns the current client constraints attached to the proxy. The getServerConstraints method returns the server's constraints for a particular remote method implemented by the proxy.

As explained above, the constraints can include, for example, integrity, anonymity, mutual authentication, delegation, and confidentiality. When the verifyProxyTrust method determines that the proxy will enforce the client's constraints by correctly implementing the RemoteSecurity interface, the proxy is "trusted."

Figures 4A and 4B are flow charts showing the steps of the verifyProxyTrust method. When a client calls verifyProxyTrust, the call has as its parameters the proxy and the client's security constraints.

As shown in steps 402 to 410 of Figure 4A, the verifyProxyTrust method first determines whether the downloaded proxy is a secure RMI stub. A stub is one example of a proxy, created by the RMI System, that uses the RMI protocol to communicate with a service. A secure RMI stub is an instance of a class generated by the java.lang.reflect.Proxy class provided by the Java™

programming language. The methods of the Proxy Class include Proxy.isProxyClass, which returns true if it is passed a proxy class that was generated by the Proxy Class, Proxy.getInvocationHandler, which returns the invocation handler associated with the proxy instance passed as its argument, and Proxy.getProxyClass, which generates a java.lang.Class object for a proxy given a class loader and an array of interfaces. These methods and the java.lang.reflect.Proxy class are explained in further detail in the "Java™ 2 Platform, Standard Edition, v 1.3 API Specification," available at http://java.sun.com/products/jdk/1.3/docs/api/java/lang/reflect/Proxy.html, and incorporated herein by reference.

The verifyProxyTrust method calls the Proxy.isProxyClass method, passing the class of the proxy as a parameter, to determine whether the proxy is an instance of a trusted generated Proxy class (step 402). As described above, the Proxy.isProxyClass method will return true if and only if the specified class was dynamically generated to be a trusted proxy class.

If the Proxy.isProxyClass method returns true, the proxy is an instance of a trusted class, and the verifyProxyTrust method tests the invocation handler of the proxy to determine if it is an instance of a trusted class (step 404). Each secure RMI stub has an invocation handler used to invoke methods on the proxy. The verifyProxyTrust method calls the local Proxy.getInvocationHandler method, passing the proxy as a parameter. The getInvocationHandler method returns the invocation handler of the proxy, and the verifyProxyTrust method then calls the local instanceof operator to determine whether the proxy's invocation handler is an instance of a local trusted class, SecureInvocationHandler. The local trusted class SecureInvocationHandler is specified in the Security class in RMI 122. The corresponding local calls made by verifyProxyTrust are:

```
Proxy.isProxyClass(proxy.getClass())
handler = Proxy.getInvocationHandler(proxy)
handler instanceof SecureInvocationHandler
```

As explained above, these methods are provided by the java.lang.reflect.Proxy class of the Java™ programming language.

If the proxy is an instance of a trusted class and the invocation handler is secure, then each socket factory instance contained in the invocation handler is checked (step 406). A socket is an end-point to a communication path between two processes in a network. A socket factory is an object that implements a method to create a new socket. Socket factories are described in more detail at http://java.sun.com/products/jdk/1.3/docs/api/java/net/SocketImplFactory.html. The class of each socket factory instance in the proxy is compared to a local list of trusted socket factory classes.

To obtain the list of trusted socket factory classes, the verifyProxyTrust method uses a local configuration database to obtain TrustVerifier.

TrustVerifiers are provided as part of the Java™ RMI Security Subsystem in RMI system 122, and are defined as follows:

```
public interface TrustVerifier {
        boolean trustedConstraintClass(Class c);
        boolean trustedPrincipalClass(Class c);
        boolean trustedClientSocketFactoryClass(Class c);
        boolean trustedProxy(Object proxy,
                                    SecurityConstraints constraints)
                throws RemoteException;
}
```

The local configuration database contains a list of TrustVerifier instances that implement methods for testing the security of a given proxy or class. The trustedClientSocketFactoryClass method will return true if the given class is a trusted socket factory class defined in RMI 122. For each socket factory instance in the proxy, the verifyProxyTrust method calls the trustedClientSocketFactoryClass method of each TrustVerifier instance, passing the socket factory class as a parameter. If at least one trustedClientSocketFactoryClass method returns true for each socket factory, the proxy is a secure RMI stub (step 408). If the proxy is not an instance of a trusted Proxy class, or if the invocation handler is not secure, or if the socket factories are not trusted, then the proxy is not a secure RMI stub (step 410).

Before continuing with the description of figure 4A, it is important to understand the difference between a unicast and an activatable stub. An RMI stub can be either unicast or activatable, depending on how it was exported by the service it represents. A unicast stub works until its service goes down. By contrast, an activatable stub will still work after its service goes down because an activatable stub is capable of restarting the service, if necessary. Each activatable stub contains an Activation ID consisting of information needed for activating the stub's corresponding object, e.g., a remote reference to the object's activator and a unique identifier for the object. Activatable objects are explained in more detail in the Java™ 2 Platform, Standard Edition, v.1.2.2 API Specification, available at http://www.java.sun.com/products/jdk/1.2/docs/api/java/rmi/activation/package-summary.html, incorporated herein by reference.

If the proxy is activatable, i.e., the proxy contains an Activation ID, (step 412) then the verifyProxyTrust method takes additional steps to determine whether the proxy is a secure RMI activatable stub. The verifyProxyTrust method obtains an activator verifier by making a remote call, using the proxy's invocation handler, to the remote service's getActivatorVerifier method, passing the client constraints as parameters (step 414). The getActivatorVerifier method is provided in the RMI system of an activatable service to enable a client to verify a proxy's Activation ID. The call to getActivatorVerifier returns an activator verifier plus optional codebase and signer information. The activator verifier is an object received from the service containing code that implements the verifyActivatorTrust method, explained below. The codebase information is the location from which the code, i.e., the activator verifier, should have been downloaded, e.g. a uniform resource locator (URL). The signer information identifies the creator or creators of the code, i.e., the activator verifier.

Because the verifier could have been downloaded, the verifyProxyTrust method checks that the activator verifier can be trusted using the codebase and signer information (step 416). If the service returned codebase information, the verifyProxyTrust method calls a local RMIClassLoader.getClassAnnotation method, which returns the location where the activator verifier code was obtained. The verifyProxyTrust method compares that location to the codebase information from the service. If they are different, the activator verifier is not used because its code is not trusted. If one or more signers is specified by the service, the verifyProxyTrust method obtains the signers of the verifier by calling the local Class.getSigners method and compares them to the signers specified by the service. If they are different, the activator verifier is not used because its code is not trusted.

In this way, the verifyProxyTrust method uses local methods to confirm the activator verifier code, which may have been downloaded and therefore could be corrupted. If the service did not specify either codebase or signer information, the verifyProxyTrust method confirms that the activator verifier was not downloaded, i.e., that the activator verifier is local, and therefore trusted. To do this, the verifyProxyTrust method compares the classloader of the verifier's class to the context classloader of the current thread or an ancestor of the context classloader. If they are the same, then the activator verifier was not downloaded, and can be trusted.

Once the verifyProxyTrust method ensures that the activator verifier can be trusted, as described above, it extracts the Activation ID from the stub and calls the verifyActivatorTrust method of the verifier, passing the Activation ID as a parameter (step 418). The verifyActivatorTrust method will return normally if the service trusts the activation ID passed to it (step 420). If any of these calls do not return normally, i.e., throws an exception, then the proxy is not a secure RMI activatable stub (step 422), otherwise trust is established (step 424).

<u>Trusting a proxy other than a secure RMI stub</u>

If the proxy is not a secure RMI stub, the client can still establish trust in the proxy. If the proxy's class has a method with the signature

```
ProxyTrust getSecureProxy(),
```

then the verifyProxyTrust method makes the following local calls:

```
proxy2 = proxy.getSecureProxy();
verifyProxyTrust(proxy 2, constraints).
```

In this manner, the proxy's verifyProxyTrust method is called recursively to get down to a secure RMI stub. The original verifyProxyTrust, i.e., the method called by the client program, then makes a call to proxy2's getProxyVerifier method.

In response, the remote service returns a proxy verifier plus optional codebase and signer information The verifyProxyTrust method checks the proxy verifier code using the codebase and/or signer information in the same way as

LAW OFFICES
FINNEGAN, HENDERSON, FARABOW, GARRETT, & DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, D. C. 20005
202-408-4000

described above with respect to the activator verifier. Once the verifyProxyTrust method ensures that the proxy verifier can be trusted, it makes a local call to the verifyProxyTrust method of the verifier, passing the proxy as a parameter. If any of these calls do not return normally, i.e., throws an exception, then the proxy is not trusted.

If the proxy is not a secure RMI stub and does not have the getSecureProxy method, then an ordered list of TrustVerifier instances is obtained from the local configuration database, as described above. The verifyProxyTrust method calls the trustedProxy method of each TrustVerifier instance, with the proxy and client constraints as parameters. The trustedProxy method returns true if the given proxy is known to be trusted to correctly implement the RemoteSecurity interface, and false otherwise. If any method returns true, then trust is established. If none returns true, trust is not established.

Although systems and methods consistent with the present invention are described as operating in a Jini™ system using the Java™ programming language, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM. Sun, Sun Microsystems, the Sun Logo, Java™, and Jini™ trademarks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries.

Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.